NSF-CCLI Final Report

Collaborative Proposal: Problem-Based Learning of Multithreaded Programming

School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332

Principal Investigator: Hsien-Hsin S. Lee
leehs@gatech.edu Tel: (404) 894-9483 Fax: (404) 385-3137

1 Introduction

The primary objective of this research project is to revamp our current computer engineering curriculum at Georgia Tech in order to fulfill the current development trend in several communities including computer architecture, system software, programming language, algorithm design, and emerging applications. Given the advent of multi-core computing in both general purpose computing as well as domain-specific computing, the inter-disciplinary trends have created profound impact to the curriculum for computer science and computer engineering majors. On the other hand, PI Lee was appointed to chairing the area committee for Computer Engineering undergraduate program by the School of Electrical and Computer Engineering at Georgia Tech in 2010. In this capacity, his primary responsibility is to overhaul the current more-than-a-decade-old Computer Engineering undergraduate program at Georgia Tech and together with his fellow committee members to propose a new Computer Engineering program for the 21st century. For his role, the learning, finding, and implementation of this project did provide a lot of insight to the curriculum revision.

Given the device scaling continues to follow Dennard's Law as well as Moore's Law, the number of transistors crammed into a single chip is steadily increasing. Nonetheless, due to several challanges including the phyiscal limitations of power density and thermal dissipation mechanism, cost of design verification, time-to-market pressure, etc., the processor designers have made fundamental paradigm shift by abandoning the pursuit of ever-higher frequency, instead, designing simpler, less-power-consuming processing cores and stamping multiple replicates on a single die. Such a design paradigm shift accelerates the production and adoption of generic general-purpose multi-core processors as well as the general-purpose graphic processors (GPGPU), making parallel computing available to everyone at the budget of desktop and even mobile computing platforms. In fact, none of the behemoth companies such as Intel, AMD, or IBM offers any single-core processor in their main CPU product. The GPGPU delivered by Nvidia and AMD-ATI essentially comprises a large number of streaming processors. The embedded system domain has picked up multiprocessor system-on-chip (MPSoC) for more than a decade. Even the ARM core contains multiple cores within a single processor IP. The amount of computing power on a modern processor is unprecedented and is completely affordable by any standard. Nonetheless, software industry has yet to make a complete transition for taking advantages of these multi- or many-core platforms due to several reasons. First of all, many applications were written and developed for single-core machines. However, more importantly if we examined a little bit further, it is not hard to realize that more than 95% of the programmers were trained to write (highly optimized) sequential codes and were not well prepared to *think parallel* when developing their applications for the now ubiquitous multi-core processors. (The quantitative data point was cited by an Intel director at Intel Labs.) As a result, the available maximal computing power is way under-utilized in certain scenarios. The primary culprit is that the software part of the computing realm is lagging far behind the advancement made on the hardware side, a huge disparity between these two camps. Inevitably, there

is a strong push from the processor industries to the academia calling for undergraduate curriculum revision (and innovation) in computer science and engineering by introducing the concepts and principles of parallel programming at the very early stage. The vision is that parallel programming will be an indispensable part and the standard for future computer programming classes. This research, with the same vision, is aimed to address this urgent need based on Problem-Based Learning (PBL) techniques, a pedagogical technique to inspiring students to solve problems through collaborative learning and hands-on experiences. The instructor in such learning environment will only provide guidance, hint, and suggestions instead of pouring the solutions directly to the students. To experience what students will experience in these PBL scenarios, the PI and his graduate students developed related techniques, examples, materials, and projects (both entry-level and sophisticated ones) to be experimented in helping students to learn how to think parallel, how to break up a sequential problem, and how to map and place these concepts into programming practice.

2 Research Activities

In the first year of this endeavor, we started to investigate multithreaded programming and approach parallization challenges by choosing two different, contempory hardware platforms. The first one is the widely available x86-based general purpose multi-core processor system, which is currently the default for all portable, desktop, and server processors. The second one we chose to investigate is the emerging platform now commonly used for scientific computing applications in addition to its original purpose for graphics rendering — general-purpose graphics processor unit (GPGPU) which by nature supports a large number of hardware threads through the capability of their internal streaming processors. Note that there are many other potential systems such as IBM Cell processors (Playstation 3 or Cell blade servers, both we at Georgia Tech have access to). But given this research is aimed at improving education and learning experiences for entry-level undergraduate students, it will be easier to design problems and have students program on these two popular desktop platforms for preparing their PBL experiences. In reality, the demand for parallelizing these more popular platforms is also growing. In the second (half) year of the project, we continued to design the lab components (or lablets) for parallel programming using these two platforms and their respective commercial or open source programming languages and tools. By gaining hands-on experiences with respect to how to perform parallel programming on these two types of systems, the fundamental parallel programming principles required will be covered for most of the common parallel platforms.

2.1 General-Purpose Multi-Core Parallel Programming

There are a large number of parallel languages, interface, API, and libraries for parallel programming on x86-based general purposed multi-core processors. New effort toward unifying and standardizing these tools (e.g., OpenCL) is under way. For learning how to program these machines in parallel manner, we chose to use OpenMP as the programming interface, and use Intel's C/C++ compiler on a Redhat Enterprise Linux machine running on the dual-socket Quad-core Intel Xeon processor platform. The choice of these tools were based the following rationale: (1) the accessibility of the toolkit, (2) the supporting commercially available tools from Intel such as Thread Checker, Thread Building Block (TBB), Parallel Studio, and (3) the familiarity of languages (C/C++) to the students, therefore, students do not need to learn a brand new language from scratch and will be able to put more concentration on how to break up and parallelize an algorithm.

For the application, as in our original proposal, we started with a simple program *wc*, a Unix utility program that counts the number of words for given input text files. The algorithm implementation of this application is by default sequential. We consider this is a perfect yet simple enough example for students to analyze the algorithm and find out strategies with respect to how to break it apart for parallelization. Most likely, this type of applications will be used as the starting point for students' first parallel programming lab

in PBL.

One obvious way to parallelize *wc* is to delegate each input text file to an OpenMP thread when there are more than one file given to the *wc* utility, in other words, implementing task-level parallelization. This type of parallelization strategy is easy to grasp for students who had no prior parallel programming experiences. However, the more interesting case we would like students to learn is the case when there is only one input file. Under this circumstance, students have to learn how to divide up the sequential execution model without running into any dependency hazard causing incorrect results. One can consider this is a classical **MapReduce** problem, which has been applied in many Google applications including their reverse indexing scheme. From this thought process, students will learn the basic concept of parallel programming, i.e., divide-and-conquer and then merge/combine the individual results into one final result. The principle of MapReduce basically is about how to **map** a sequential model into independent sub-tasks, the main part of parallelization, and then **reduce** these individually produced results into one final answer. Toward this for *wc* utility under the instructor's guidance, students should find that the input buffer can be sub-divided into almost equal-sized chunks, but to keep the integrity of words, the subdivision must be performed in a way that all the divvied-up chunks must end upon a word boundary, a correctness requirement. Therefore, the parallelized version will not count the same word twice. Through this example, students will learn this basic MapReduce concept for parallel programming and use OpenMP to parallelize the code.

In addition, we would also have students develop parallel programs for more complex algorithms using OpenMP after they become more efficient in handling parallelization as well as the API itself. In this context, a 3D medical image reconstruction algorithm based on computing tomography (CT) was investigated and studied to create the learning experiences. We studied the algorithm proposed by Alex Katsevich, the first theoretically exact cone beam image reconstruction algorithm for a helical scanning path in CT. We studied the parallelization strategy and had it parallelize at a very fine-grained to be able to take advantages of both single-instruction multiple-data (SIMD) SSE instructions provided by x86 instruction set as well as to execute these SIMD streams on multiple processor cores in a general purpose multicore processor. This exercise involves the dissection of the algorithm to isolate independent computation at fine-grained level, understanding the shared data structure, and the implementation of both SSE at instruction level and OpenMP at the task level. Through analyzing this problem, students will be able to follow the same methodology for parallelizing a complex real-life computing problem. More details will be given in the next section.

2.2 Parallel Programming on GPGPU

We chose GPGPU as another pedagogical tool for parallel programming given its popularity and high computation efficiency and cost-effectiveness. The objective of this exercise is for students to learn how to parallelize a program on a massively parallel machine. Such platforms used to exist only in large-scale domain back in the 90s (at a prohibitively high cost), but now we have such platform accessible to everyone on their desktop or even mobile computers. To begin with, the applications running on such platform must be parallel-friendly, in other words, their algorithms must be inherently data-parallel. We expect that through this effort, students will learn the drastic differences of parallelizing applications on a general purpose Intel processor versus on a GPGPU that supports data-level parallelism. They will also learn the differences in their programming productivity. We also examined the OpenCL, a standard for making heterogenenous computing easier, but did not have enough time and budget to pursue it further at the time of the project closing. However, we anticipate to provide OpenCL based programming exercises (lablets) in the future.

The platform we chose for students to use is Nvidia's G80 and Tesla C870. Students will use CUDA (Compute Unified Device Architecture), a parallel intrinsic-like interface developed by Nvidia to write their parallel programs on these platforms. Architecturally speaking, Nvidia's GPGPU is a *many-core* processor. However, each core (also known as a shader unit) was designed to excel in performing concurrent graphics

operations or parallel SIMD-type of computation very efficiently rather than dealing with sequential tasks.

The application we chose for parallelizing is the same 3D medical image reconstruction algorithm designed by Alex Katsevich algorithm used in CT scanner as mentioned in the previous section. This application is data parallel inherently. But to run it efficiently on GPGPU, programmers require some delicate considerations. In addition, once students become proficient in parallelizing simple code on IA-based multi-core system, they will be asked to try to parallelize the Katsevich algorithm on an IA-based multi-core platform and compare their productivity and performance results against their parallelized CUDA version running solely on a GPGPU.

To parallelize this algorithm on Nvidia's G80 and Tesla C870, we design the lablets to let students accomplish the following tasks through their thought process in PBL: (1) to recognize the independent portions that can be safely parallelized without compromising the correctness of computation and (2) to recognize the steps that cooperatively update the same piece of data. To gain insight for the above points, students have to analyze the structure of the Katsevich algorithm and set up their parallelization strategies. Some interesting aspect of parallel programming we want students to learn through this process is — in lieu of waiting for dependent yet simple, repeated data to be communicated, it is sometimes faster to just re-calculate the data on each individual core by exploiting its computing power. This highlights the common performance and scalability bottleneck of a parallel system: the cost of synchronization and communication. Once students grasp this concept in mind, certain part of an algorithm could become more parallelizable. The principle here is to trade communication off with additional computation. Since computation is cheaper (in terms of performance) than communication in a today's massively parallel system, eliminating communication and replacing them with additional computation is an acceptable strategy to gain performance.

One reason we chose to let students work on a GPGPU is to have them learn the architecture of massively parallel machines as well. People used to say *"Parallelizing code is easy if you don't care about performance."* That implies, it won't be easy to have programmers learn how to optimize applications without knowing the architecture of the underlying machine. In particular, for GPGPU, students have to deal with limited shared memory issue.

2.3 Material Integration into Georgia Tech's Computer Engineering Program

The final goal of this research effort is to integrate what was investigated and designed (such as programming exercises and labs) into our curriculum. Fortunately, Georgia Tech is undergoing a major curriculum revision for our computer engineering undergraduate program since 1999. No timing is more perfect than now to have our PBL research results to be integrated into our brand new curriculum. Moreover, PI Lee was appointed to chair the curriculum subcommittee for computer engineering program, therefore, his role gives a lot of leverage to turn their research concept into practice in curriculum. In essence, the design flow of PBL multithreaded programming including parallelization stratgy, parallelization tools, and parallelization methodology will be used as a gudie in the new courses that discuss concurrency. More information will be discussed in Section 3.2.

3 Major Findings and Outcome

3.1 Course Materials

For the wc example on IA Multi-core platform, the most critical task students need to do after parallelizing their code is to verify the results. This is typically the most tedious and time-consuming part of parallel programming. During the good old days, debugging parallel code on a large-scale parallel machines is one of the most unpleasant engineering processes, due much to the non-deterministic nature of parallel programs and lack of parallel debugging tools. One reason we chose to use the Intel platform is because there are now many supporting tools such as Thread Checker to help diagnosing and debugging code written in, e.g.,

OpenMP. Potential data race issues in the program are red-flagged by the tool itself, making debugging possible and improving the programming productivity.

Once verified, the next step for students to learn is how much performance improvement was achieved in their parallel version (e.g., wc utility.) This is also a great educational opportunity for students to understand the crust of *Amdahl's Law*, i.e., scalability can be significantly hindered by the sequential part of a parallel code. Another advantage to use Intel platform is the several options of compilers. Students can learn from the development experiences about how a good compiler or compiler optimization knobs play a role in delivering high performance. In addition, students can vary the number of threads in the OpenMP directive and experience the variation in speedup. They are likely to find out that more threads do not necessarily reduce the total execution time due to growing inter-core communication overhead. At Georgia Tech, we have developed our own OpenMP parallel version of wc and measured the speedup of the program. We also analyzed the cache behavior of our workloads to answer those more in-depth questions about performance slowdown when we scale up the input sizes.

For the Katsevich image reconstruction algorithm, verification could be done by comparing the results generated by GPGPU with those generated by the IA-based multicore processors. Here, we expect that students will have some interesting learning from the comparison. They will likely find the results from these two platforms are not bit-by-bit the same but within a negligible margin. From that, they will learn and understand the precision issue of floating-point implementations for different systems. Finally, we want them to compare the performance of the same algorithm parallelized on different machines and analyze why one is faster than the other, and what are the performance bottleneck, and if there is anything that can further be optimized to close the gap.

3.2 Computer Engineering Curriculum Revision at Georgia Tech

PI Lee, as mentioned earlier, was chairing the undergraduate curriculumn committee for Computer Engineering program at the School of Electrical and Computer Engineering at Georgia Tech in 2011. The objective of this activity is to reconsider the requirement for computer engineering major at the School of Electrical and Computer Engineering at Georgia Tech and to provide those graduates competitiveness in the 21st century. There is a common awareness of two major paradigm shifts for computer engineering major: (1) multi-core and many-core computing and (2) computer engineering has become more software-centric rather than hardware-centric due in part to (1). For the curriculum designed in the 80s and 90s, most of the curriculum deisgns focus on the hardware design aspects such as microarchitecture, pipelining, VLSI design, CAD tools, RTL development and synthesis, and so on. However, by looking at the job opportunities for computer engineering major today, a slew of demand, in fact, came from embedded software development, system integration, and analysis for computing efficiency (by considering performance, energy, etc.) Even though they are more software-oriented, the skills required are a result of hardware advancement. For example, a computer engineering major may not need to design cache memory, but they need to understand the cache memory design well in order to optimize their embedded programs running on mobile devices. In other words, the new computer engineering graduates will be less likely to design these hardware from scratch themselves but they still need to understand these hardware in order to carry out their tasks, *i.e.*, the application of such knowledge is drastically different from decades ago. This becomes the goal of our curriculum revision to provide necessary training in order to satisfy the realistic demand.

To fulfill this mission, our new curriculum proposal was designed to center around two timely concepts for modern computer engineering major: *concurrency* and *energy*. The curriculum was also completely overhauled to align with these two concepts. The portion related to this project is the concurrency part. We will integrate our proglem-based learning (PBL) lablets investigated and designed in this research to three main courses in this new curriculum including one freshmen level course "Computing for Engineers",

one sophmore level course “Engineering Software”, a junior level “Architecture, Concurrency, and Energy” course. These courses contain lab hours for practicing our PBL method. First, basic parallelization concepts will be introduced into the freshmen course, such as how to parallelize simple algorithms such as wordcount, sorting algorithm, data structure representation using MatLab. Then in the sophmore course, students will be introduced the concept of threads and concurrency using high level language and perform programming assignments to write parallel programs including lock-based programming and learn pthread and OpenMP programming. The junior level course will discuss concurrency at different levels from the execution standpoint including instruction-level parallelism, thread-level parallelism, data-level pararllelism, speculative multi-threading. As such, the students will be able to bridge the actual machine execution model with the programming models they learned from previous pre-requisite classes.

4 Training and Development

The research activities of this project are to define and design the PBL examples, lablets, tools, and materials. The graduate student who was supported by this project went through the process to tackle the problem and implemented the parallel applications we opted for and learned from the actual experiences. We developed the parallel versions of our target code using OpenMP for IA-based processors and CUDA for Nvidia’s GPGPU. The final goal is to apply these applications (or lablets) onto different parallel languages or programing APIs for these two main stream platforms in our newly revisited computer engineering curriculum.

5 Contributions

The accelerating advent of multi-core and general-purpose many-core graphic processors made parallel processing available to everyone at the desktop budget. The amount of computing power on a contemporary single processor die is unprecedented. Nonetheless, there is a reality that a majority of the available software were mostly developed for single-core machines and continue to run on today’s dual- or quad-core platforms due to several reasons. First of all, many of them were written and developed on single-core machines. However, more importantly if we examined a little bit further, it is not hard to realize that more than 95% of the programmers were trained to write (highly optimized) sequential codes and were not well prepared to think parallel when developing their applications for the now omnipresent multicore processors. As a result, the available maximal computing power may be way under-utilized in certain scenarios. The primary culprit is that the software part of the computing realm is lagging far behind the advancement made on the hardware side, a huge disparity between these two developers. There is a strong urge from industries to academia calling for undergraduate curriculum renovation (and innovation) in computer science and engineering by introducing concepts and principles of parallel programming at the very early stage. The vision is that parallel programming will be an indispensable part and the standard for future computer programming classes. This research, with the same vision, is aimed to address this need based on Problem-Based Learning (PBL) techniques, a pedagogical technique to inspiring students to solve problems through collaborative learning and hands-on experiences. The instructor in such learning environment will only provide guidance, hint, and suggestions instead of pouring the solutions directly to the students. To experience what students will experience in these PBL scenarios, the PI and his graduate students are developing related techniques, examples, materials, and projects (both entry-level and sophisticated ones) to help students (mostly undergraduates) to learn how to think parallel, how to break up a sequential problem, and how to map and place these concepts into programming practice.

6 Contributions to Other Disciplines

The outcome of this work can be applied to other non-EECS major who need to learn parallel programming in their discipline. For example, aerospace engineers who design and simulate their aircraft design

using large-scale parallel machines, chemists, material scientists, or biologist who would like to efficiently parallelize their chemical compound or protein folding algorithms or to develop new drugs, meteorologist who use parallel machines to simulate and study ocean circulation and climate changes, or radiologists who want to accelerate the image processing of computing tomography like the one we studied on GPGPU, to name a few. Since the PBL learning will be neutral to the understanding of specific algorithms. The method will be applicable to those who are desired to learn parallel programming for their own specific subject of interests. Moreover, our courses in the new curriculum are intended for a larger scope. Historically, a few computer engieering courses were quite popular and taken by students who major in other disciplines including mechanical engieering, industrial and system engineering, biomedical engineering, chemical engineering, biology, etc. It is the PI's firm belief that the new classes dealing with concurrency will attract many students from these inter-disciplinary areas for learning multithread, multicore programming for their own applications of interest.

7 Contributions to Resources for Research and Education

7.1 Scholarly Contribution

The project activities undertaken and the findings including code base, parallelization strategy and algorithms, as well as the comparative studies of different parallel platforms were documented and appeared as a book chapter [1] in a new book GPU Computing Gems published by Morgan Kaufman Publishers in 2011. In addition, our source codes were released to a couple of other researchers from academia who made requests for accessing and analyzing our OpenMP and CUDA source code. Through the distribution of the book and source code, our parallelization experiences and methods will have a more profound impact to the research and pedagogical community in multithreading programming for both general purpose processors and specialized accelerators.

7.2 Human Resources

Throughout the project period at the Georgia Tech site, we had successfully trained two graduate students (Adberrahim Ali Benquassmi and Eric Fontaine). A Master's degree was awarded to Adberrahim (Ali) Benquassmi who was sponsored by this project for one and half a year and was the main contributor for the CUDA parallel programming for the 3D image reconstruction Katsevich algorithm (while Eric Fontaine focused on the same algorithm but parallelized it on a generic multi-core processor using OpenMP.) Mr. Benquassmi graduated in 2010 and has joined Garmin International in California as a software engineering working on the next generation 3D graphics rendering algorithm for the global position system (GPS) display embedded inside automobiles. The job responsbility is very close what he was trained for during this period of project.

References

[1] Abderrahim Benquassmi, Eric Fontaine, and Hsien-Hsin S. Lee. Parallelization of Katsevich CT Image Reconstruction Algorithm on Generic Multi-Core Processors and GPGPU. *Chapter 41, GPU Computing GEMS, ed. Wen-Mei Hwu. Morgan Kaufmann Publishers*, 2011.